Amendments to the Specification:

Please replace the paragraph beginning at page 4, line 25 as with the following amended paragraph:

FIG. 2 illustrates a method 200 for client process 120 to request a service from remote server process 150. The method begins when client process 120 and remote server process 150 are initialized (step 210) – for example, when a user launches a software application program or programs implementing the respective processes. Client process 120 generates one or more data values, which may be, e.g., input parameter values (e.g., particular integer, real number, string, array, pointer or other structure) required by remote server process 150 (step 220), and stores the data values in the address space of client process 120 – for example, adding the data values to a memory stack in client process 120's address space. At runtime, system 100 calls a type description function in library 170 to obtain a type object describing the data type of the data values (step 230). As used in this specification, runtime "runtime" is used in its conventional sense to refer to any of the time during which a program is executing and is not limited, for example, to time immediately ~~refers~~ before client process 120 requests the service from remote server process 150. Thus, as used herein calling the type description function at runtime includes, for example and without limitation, calling the function upon initialization of client process 120 or remote server process 150, or during execution of client process 120, either before or after generation of the required data values. Either client process 120 or remote server process 150, or both, may call the type description function in library 170. If the processes are physically remote from each other, they can use distinct copies of the library 170. In one implementation, both client process 120 and remote server process 150 call the type description function in library 170, creating two instances of the type object – one on the client side and one on the server side – which will be invoked by the respective processes, and which can be identified by a common registration [[id]] ID agreed upon by the two processes. Alternatively, the process that calls the type description function in step 230 can send the resulting type object or objects to the other process at some time before the data is to be sent or with the data. Optionally, system 100 verifies that client process 120 and remote server process 150 agree on a

common type description for the data type -- for example, by causing client process 120 to send a "checksum" of the type object, generating an error message if the two objects do not agree.

Please replace the paragraph beginning at page 6, line 16 as with the following amended paragraph:

Likewise, library 170 can include functions that return lcType objects describing more complex data types, such as strings, arrays, pointers and structures. A lcTypeCStringCreate function, for example, takes an integer representing a maximum string length (maxLength) as an input parameter and returns a lcType object describing a cstring having the specified maximum length. When a data value of this type is to be encoded or decoded in system 100, the maxLength parameter determines the number of bytes required to encode the string's length and also places a limit on the size of string the decode method will attempt to allocate and fill. During encoding or decoding, any string longer than maxLength (or any array, structure or other data value that exceeds predetermined size limits for its defined type, as discussed below) will cause an error that will abort the entire encode/decode hierarchy. Client process 120 can be configured to refuse to encode any string (or other data) that exceeds its predetermined size limit. Similarly, remote server process 150 can reject such data and identify a process sending such data as faulty or malicious and initiate appropriate action by a user or by system 100.